UCRL-MI-203744

Attachment 1

Statement of Work for the Open Source Parallel Performance Measurement Package

National Nuclear Security Administration
Advanced Simulation and Computing
PathForward Program

April 13, 2004

Contents

INTRODUCTION

The purpose of this Statement of Work is to describe an open source development effort to produce a Parallel Performance Measurement Package (PPMP) generally aimed at the needs of the High Performance Technical Computing community and, in particular, the needs of the National Nuclear Security Administration (NNSA) and the Advanced Simulation and Computing (ASC) Program parallel capacity computing systems. The University requires an open source software tool, or suite of integrated open source software tools, with a common look and feel, that will provide a collection of basic measurements to give insight into the performance of parallel programs.

1 REQUIREMENTS

The specific Mandatory Requirements the software shall meet are delineated with a "MR" designation. Software technical Target Requirements that are highly desirable, but not mandatory, are delineated with a "TR-1" designation.

In addition to the Mandatory Requirements, the Offeror may propose any Target Requirements for the software, and any additional features consistent with the objectives of this project and the Offeror's project plan that the Offeror believes will be of benefit to the University. Target Requirements and additional features proposed by the successful Offeror may be included in the resulting Subcontract.

1.1 Software Functionality: Applications to be Measured

1.1.1 The software shall report performance for parallel applications written in a combination of languages (at least Fortran90, C++, C). Performance results shall be available from the executable file with no recompilation or relink required. If the application has been compiled with symbol tables, trace back to source shall be available. The software shall work with parallel applications using the Message Passing Interface (MPI) communication library. The software shall also work with single task applications that do not involve MPI. [MR]

1.1.2 The software performing measurements shall function in both interactive and batch modes with a choice of Graphic User Interface (GUI) or command line interfaces when interactive mode. [MR]

1.1.3 The software should work with applications that invoke dynamic-linked libraries. [TR-1]

1.1.4 The software should work with optimized object files. Describe any limitations imposed by compiler optimization. [TR-1]

1.1.5 The software should work with applications that use OpenMP directives for thread parallelism. [TR-1]

1.1.6 The software should be able to attach to an already running job to collect performance measurements. [TR-1]

1.1.7 Use of the software should be scriptable so that it can be invoked from other tools. [TR-1]

1.2 Software Functionality – the Performance Measurements and Reports Available

1.2.1 Basic application performance reports available shall support results from hardware counters applied at function level, time profiling at the function level, profiling from program counter sampling, and statistics from MPI usage by call type. [MR]

1.2.2 Performance measurement reports shall be available both in interactive GUI format and in text (see 1.2.7) formats. The interactive GUI format should include a display to tie the measurements back to source code. The text formats are especially intended to facilitate, but not be limited to, collection of measurements from batch runs of the application. [MR]

1.2.3 The software should support performance measurements for MPI call tracing, statistics from I/O (input/output) interfaces, I/O call tracing, thread specific profile and hardware counter data, heap and memory consumption statistics. [TR-1]

1.2.4 Statistics reported at the function level should include data that is both inclusive and exclusive of functions called from within the function. [TR-1]

1.2.5 Statistics at a finer grain, e.g. object, block or line level should be possible. Describe what mechanisms are available for this. [TR-1]

1.2.6 For some statistics, such as MPI call usage, a multi-level function or object traceback should be available to distinguish between various invocations. Describe which of the measurements provided will include this feature. [TR-1]

1.2.7 Text reports of measurement results should be available with mark-up-language commands imbedded, e.g., HTML, to facilitate readability and embedding graphics. In addition, or alternatively, the results might be available in a commonly available spreadsheet or database format. Describe your approach for various kinds of measurements. [TR-1]

1.2.8 Other measurements that tie to event-based data, such as timers or interrupts based on hardware counter thresholds, are also of interest. Describe any functionality supported. [TR-1]

1.3 Open Source Software

1.3.1 All software developed under this Subcontract shall be released under a University approved Open Source license (see the Sample Subcontract) and delivered to the University with source code. [MR]

1.3.2 End user documentation, component interface documentation, application programming interface (API) documentation and software test suites should be provided as part of the open source release. The methods used for documentation

and testing should be broadly available—not requiring proprietary products with limited access. [TR-1]

1.3.3 Read/Write access to the software source repository should be provided to at least one person, designated by the University, at each of LLNL, LANL, and SNL. Read access to the software should be broadly available. Describe the software gatekeeper plans. [TR-1]

1.4 Operating Systems and Hardware Environments Supported

1.4.1 The software shall be deployed and tested on the Intel Product Family including Pentium 4, IA-64, and X86-64 architectures. The software shall also be tested on the AMD Opteron-based architecture in both 32 and 64 bit modes. [MR]

1.4.2 The software shall measure performance for applications built with GNU compilers and running on parallel Linux clusters. Support for clusters using an OS image per node (with or without disks) and clusters using a single system image model, specifically BProc Linux, shall be required. [MR]

1.4.3 The software should measure performance for applications built with other vendor-supported compilers, specifically the Intel compiler suite (Intel C/C++ and Intel Fortran on the Pentium 4 and IA64 architectures. For the Opeteron architecture, the compilers of interest are Portland Group and the Absoft and Lahey Fortran compilers. [TR-1]

1.4.4 It should be routine to use the software to measure performance on jobs of up to 256 tasks. The time to initiate a performance measurement for 256 MPI tasks of one of the demonstration codes should not exceed two minutes longer than the time to initiate the application without the measurement. [TR-1]

1.4.5 It should be possible to use the software to measure performance on jobs involving thousands or tens of thousands of tasks/threads. Describe how the software will address very large parallelism. Which measurements will be supported at large scale? Which measurements will require special approaches? An example of an acceptable approach to large-scale parallel jobs is to involve some subset mechanism to limit volume of data or make GUI display tractable. [TR-1]

1.5 Extensibility and Usability of Design

1.5.1 Documentation specifications of system interfaces and extensibility interfaces shall be provided to enable extending the work to other systems. [MR]

1.5.2 New performance measurement features should be easy to add, both by the Offeror and by other collaborators working with the software. Describe the software architecture that will enable this. Examples are a measurement component architecture, a stack unwind library, program counter recording library, source line number and file name lookup library. [TR-1]

1.5.3 The Offeror should provide an interface usability testing plan that will involve tests with University-designated users on real applications before the interfaces are finalized. [TR-1]

1.5.4 The software should incorporate an approach to generation of trace files that will help control and minimize the size of files generated. At the same time, the technique should preserve the ability to read and initialize the analysis of the trace data in less than five minutes. [TR-1]

1.5.5 To facilitate portability of software to other platforms, the Performance Application Programming Interface (PAPI) to the hardware counters should be supported. [TR-1]

1.5.6 To facilitate the portability to Light-Weight-Kernel operating systems, the offeror should describe the software issues that will occur in an environment where there are no threads or sockets available on the nodes where the target applications run. [TR-1]

1.5.7 Some target applications may involve hundreds of thousands of lines of source code, with thousands of functions, in hundreds of files, stored in tens of subdirectories. The software should support mechanisms to allow the user a "scalable" GUI for specification of what to measure. Similarly any GUI for viewing results needs to be designed in a way that is manageable for the large applications. For example, scrolling through an alphabetic list of all the function names in the application is not acceptable. Applications that involve "mangled" symbol names should present the names in a form recognizable by the user—not names that are hundreds of characters long. [TR-1]

1.5.8 There should be mechanisms to establish initial defaults and preferences. In particular, it should not be required that the user enter long pathnames to identify target application directories over and over. There should be a way to save this state from one measurement experiment to the next. [TR-1]

1.5.9 The GUI should support "cut and paste" functionality—especially where non-default pathnames are to be specified. [TR-1]

1.6 Software Maintenance and Support

1.6.1 The Offeror shall continue serving as the open source gatekeeper, maintain all software documentation, and provide first line bug support for a period of two years after initial deployment of the software. [MR]

1.6.2 The Offeror shall provide enhancements as required by the University Technical Representative for a period of two years after initial deployment of the software. [MR]

1.6.3 The Offeror shall continue efforts in the productization of the open source package for a period of two years after initial deployment of the software. [MR]